(8)

(a) $100 < \log[\log n] < \log n < \log(n) < n <$

$\log(n!) = n\log n < n^2 < 2^n < 2^{2n} < n^n < n!$

(b) $1 < \log(\log n) < \sqrt{\log n} < \log n . 2 . \log n < \log 2n < n$

$= 2n = (1n < n \log n = \log(n!) < n^2 <$

$2(2^n) < n!$

(c) $96 < \log p^n < \log_2 n < 5n < n \log_4 m < n \log_2$

$= \log(n!) < 8n^2 < 7n^3 < 8^{2n} (n!$

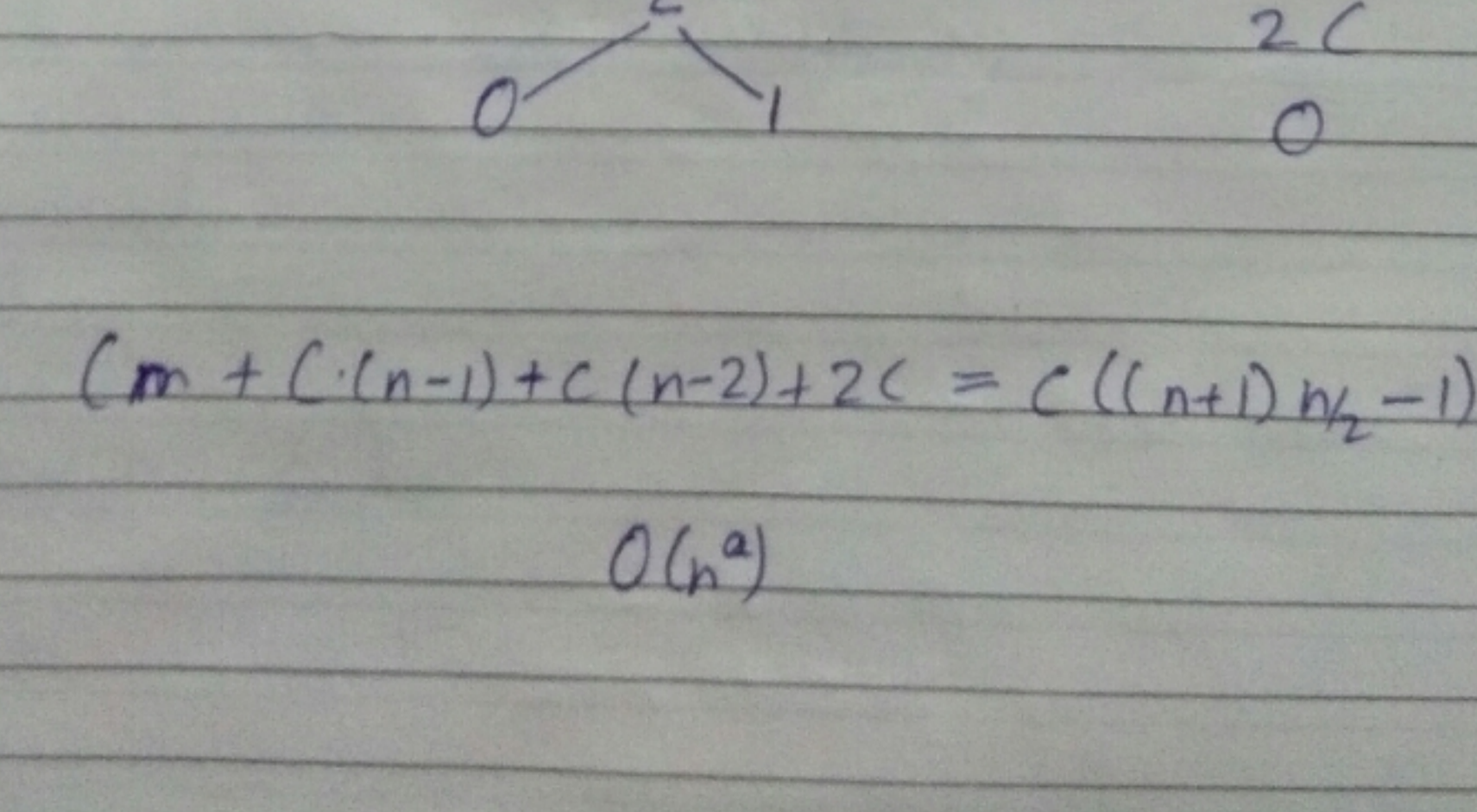
⑥ for (int i=2; i<=n; i = pow(i,K))
{
}
where K is a constant

T.C = O(log(log n))

⑦ Recursion relation will be

T(n) = T(0.n(10) + T(n/10) + O(n)

Subproblem sign — Total partitionating time



$Cn + C\cdot(n-1) + C(n-2) + 2C = C((n+1)n/2 - 1)$

$O(n^2)$

```
i=i+2;
while (j<m)
{
    count++;
    j+=i;
}
```

④ $T(n) = T(n/4) + T(n/2) + cn^2$

$T(n/2 \mid) = T(n/4)$

equation can be rewritten

$T(n) = 2T(n/2) + cn^2$

using mask's method.

$a=2, b=2$

$c = \log_2^2 = 1$

$n^c = n$

$f(n) = n^2$

$f(n) > n^c$

$T.C = \theta(f(n)) = O(n^2)$

⑤
```
int fun(int n)
{
    for (int i=0; i<=n; i++)
    {
        for (int j=1; j<m; j+=i)
        {
        }
    }
```

$TC = O(m^2)$

③ Program having complexity

(i) n (logn)

```
int sum = 0, i, j;
for (i=1; i<=n; i*=2)
{
for (i=1; i<=n; i++)
  {
    sum+=j;
  }
}
```

ii) n^{3^2}

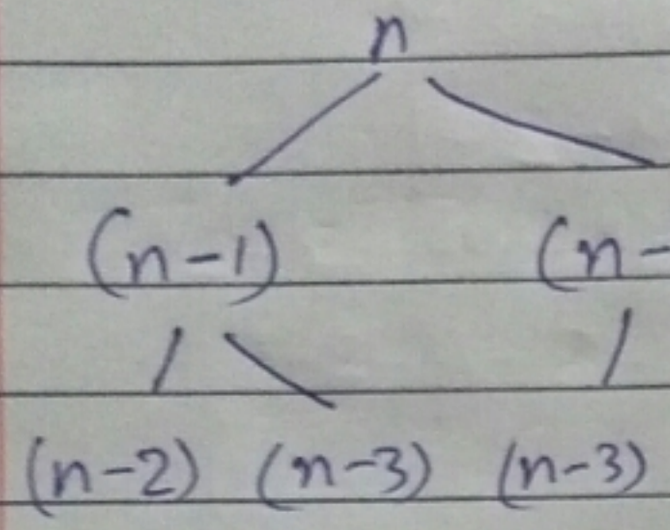
```
int i, j, K  sum=0;
for (i=0; i<n; i++)
  {
    for (i=0; i<n; i++)
      {
        for( K=0; K<n; K++)
        {
          sum+=K;
        }
      }
  }
```

(iii) log (logn)

```
int  i=2, count=0, i;
```

② Recurrence relation for recursive function that prints fibonacci series.

$$T(n) = T(n-1) + T(n-2) + 1$$



$$T = 1 + 2 + 4 + \cdots + 2^n$$

$$a = 1 \qquad r = \frac{2}{1} = 2$$

$$\frac{a(r^{n+1}-1)}{r-1} = \frac{1(2^{n+1}-1)}{2-1} = 2^{n+1} - 1$$

$$O(2^{n+1}) = O(2^n \cdot 2)$$

$$T.C = O(2^n)$$

Space complexity of it is $O(n)$ be cause there are 'n' no of function and calls during recursion. without using recursive is will $O(1)$

DAA Tutorial - 2

NAME - TARUN KUMAR DHIMAN
ROLL NO. - 1918917
Sec - IT

①
```
void fun(int n)
{
    int j=1, i=0;
    while (i<n)
    {
        i = i+1;
        i++;
    }
}
```

'i' can be defined according to the relation $i_j = i_{j-1} + j$. The value of j increase by one by for each insertion is the sum of first 'j' position insert integer. If K is the tot a no of iteration taken by program, the while loop terminate if

$$1+2+3+\cdots\cdots+K = \left[\frac{K[K+1]}{2}\right] > n \quad \text{so} \quad K = O(\sqrt{n})$$

$$T.C = O(\sqrt{n})$$